A Runtime Customizable Access Control Model and Its Use in a Garment Design Cooperation System

Ran TAO, Ze-ping LV*, Xin LUO, You-qun SHI, Ming ZHU and Jie ZHANG
School of Computer Science and Technology, Donghua University, China
*Corresponding author

Keywords: Access control, Customizable, Rights management, Dynamic authorization.

Abstract. At present, many custom software systems generally adopt role-based access control. In such systems, a specific user may be authorized to one or several roles, and each role and its permissions are pre-defined at the beginning of the system design. It is difficult for managers to dynamically define new roles for users with personalized permissions during system runtime after system delivery. This paper proposes a runtime customizable access control model for the custom-built cooperation software system which has many access objects and fewer users, each user has different access requirements, and each user’s requirements are uncertainty and can only be confirmed after a cooperation project has been launched. To verify the validity of the model, a garment design cooperation system with this access control model was implemented as a case study. Experiments show that with the use of this model, managers can dynamically create personalized roles and authorize them to different project members when a new cooperation project is launching. The model and the case can help researchers and developers to design and develop their own runtime customizable access control model and custom-built cooperation software system.

Introduction

In a society where information technology is developing at a high speed, various information application management systems have being sprung up. The information management system is a complex human-computer interaction system, it is important to build a secure and stable access control system to ensure its security. Access control plays an important role in the security of the entire system. Any multi-user system inevitably involves permission requirements and needs to address security services such as data confidentiality, data integrity, and access control [1]. ISO defined five Security service functions in the design standards of network security systems: identity authentication tasks, access control services, data privacy services, data integrity services, and non-repudiation services. For example, the access control service requires the system to control which resources the operator can access and how to operate the resource based on the operating privileges set by the administrator [2].

This paper takes access control services as a research direction. By accessing the control service, access to critical resources can be restricted, and the intrusion of illegal users or unnecessary damage caused by improper operation of the user can be prevented. In order to manage knowledge within an organization or among organizations, various custom-built cooperation software systems have been developed. Such systems generally adopt role-based access control, a specific user may be authorized to one or several roles, and each role and its permissions are pre-defined at the beginning of the system design. It is hard for managers to dynamically define new roles for users with personalized permissions during system runtime after system delivery. At the same time, existing access control policies are not well addressed in some specific management software. For example, in some collaborative management collaboration software, the operation authority is large, the number of users is small, and the roles and permissions are uncertain, resulting in complex role definitions. In addition, the traditional access control model to view each user's permissions is not intuitive, not efficient and inconvenient.
For the above problems, in this paper, a universal, safe, easy of manage and customizable access control model is proposed, with the use of it, personalized roles can be dynamically created and then be authorized to various users in real time during system runtime. At the same time, the system can query and display user rights quickly and efficiently.

The rest of this paper is organized as follows: The second part summarizes related concepts and characteristics of various access control technologies. The third part proposes the design of the runtime customizable access control model. The fourth part uses a case study to verify the validate of the module. The fifth part evaluates the differences and innovations between our access control model and the existing rights management system.

**Related Literatures Review**

**Access Control Policy**

The access control policy refers to the main strategy for data resource security prevention and protection. The main purpose is to prevent illegal access or illegal use of data resources, that is, to prevent unauthorized users from accessing protected data resources, and to allow legitimate users to have access to data resources. Prevent unauthorized users from unauthorized access to protected data resources [3]. The three elements that implement access control are subject, object, and access control strategy. The subject refers to the user's autonomy in the access process of the information resource; the object refers to the object or resource accessed by the user; the access control strategy refers to establishing a rule to limit whether the subject has the right to operate the object [4].

Access control strategy plays an important role in various software development processes. It is inextricably linked to the reliability and security of software systems. The choice of access control policies and the division of access control rights in the system are software system development projects. An important issue that personnel need to face.

**Access Control Technology Model.**

Access control includes three mechanisms: integrity, validity, and confidentiality. Integrity control means preventing resources from being illegally added, deleted and modified. validity means preventing unauthorized access to users from accessing resources, inquiring, etc. confidentiality means protecting special key resources to prevent illegal access to resources. The current access control models mainly include Discretionary Access Control (DAC), The Mandatory Access Control (MAC) and Role-Base Access Control (RBAC).

DAC is a self-management of users. It is necessary to first determine the type of user group and the role of the subject, and then manage it to manage access control. The “autonomy” means that the user can delegate the authority to others according to his or her own wishes, and can also revoke the access rights of the other party. In this process, the administrator is not required to manage. Most systems implement access control based only on autonomous access control mechanisms, such as mainstream operating systems (Windows NT Server, UNIX systems), firewalls (ACLs), etc... It is precisely because in the autonomous access control, the subject can establish access rights through his own needs, resulting in a series of access rights are not constrained, which will lead to the risk of data leakage and illegal access [5-6]. Therefore, a mandatory access control model has emerged.

MAC is centralized management of users. For example, Selinux is a mandatory access control implemented by the National Security Agency (NSA) and the most prominent new security subsystem on Linux. Under the constraints of this access control system, processes can only access files that are needed in his tasks. The access user is granted different security levels, and the management system determines whether the user has the corresponding access right by comparing the security level of the access user and the data information. This requires the administrator to set the security level permission attribute for each subject and object, and the subject does not have the right to modify its own authority, so the permission setting is mandatory, the operation is cumbersome and inconvenient [7].
RBAC is a role-based access control by creating different roles and assigning different roles to the corresponding users. In RBAC, the main method of access control focuses on the creation of user roles. Since all users in the system need to obtain access control over data resources through roles, administrators need to set different role permissions according to requirements and then distribute the roles to users, roles associate users with permissions. The three principles of the RBAC model: (1) When the user of the system does not assign a role, the user does not have permission to access any resource, ensuring that any user must obtain the corresponding function through the role. (2) The roles in the system must be authorized by the administrator to ensure that the user can only obtain authorized roles. (3) If the user wants to obtain the right to execute the transaction, the transaction must authorize the user's current role [8-9].

Through the above-mentioned related literature discussion, three mainstream control models can be compared and analyzed, as shown in Table 1.

<table>
<thead>
<tr>
<th>Access control model</th>
<th>control method</th>
<th>advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC</td>
<td>Establish access control list.</td>
<td>Suitable for small information systems with few permissions, easy to view and easy to understand.</td>
<td>The grant of access rights is passable. At the same time, access rights are not constrained, which is not conducive to data confidentiality.</td>
</tr>
<tr>
<td>MAC</td>
<td>Pre-defined security levels for user and data information, mandatory allocation.</td>
<td>Safer.</td>
<td>The modification of permissions is very complicated, inconvenient, and vulnerable to system security vulnerabilities.</td>
</tr>
<tr>
<td>RBAC</td>
<td>Introduce roles to separate the permissions from the logical relationships between users.</td>
<td>Flexible configuration of permissions, easy management, easy operation, and different access control strategies.</td>
<td>Pre-privilege configuration has a large workload.</td>
</tr>
</tbody>
</table>

In Table 1, for the autonomous access control model and the mandatory access control model, there is a direct interaction between the user and the authorization. Therefore, when the number of system users fluctuates greatly or user rights need to be adjusted, it will increase the system administrator's work of changing user rights, making the operation cumbersome and inconvenient.; for the role access control model, there is no direct connection between the user and the authority, and the two are associated through the role. When a user accesses data, the user obtains the corresponding access rights through different roles. Because the role is fixed, when the system user changes on a large scale, the system only needs to adjust the user role. The model can greatly reduce the complexity and difficulty of system user rights management and can allocate the rights of the system more flexibly and quickly. For example, when a user accesses different data and needs multiple different permissions, the user can be assigned a different role to manage the rights.

**Linux Rights Management Method**

The Linux rights management method uses UGO (User, Group, Other) to set the permission bits on files and directories, thereby managing the permissions of the file system, and is used to control access to files or directories by users or groups of users [10]. The UGO rights management method divides the operators of files into file owners, groups of users, and other group users. The file owner refers to the user who created the file. It is owned by the file and can set the user's read, write and execute permissions. The same group of users refers to users in the same user group as the file owner. Other user groups are users who are not in the same user group as the file owner [11].

In a Linux system, after a file is created, it has three modes of operation: read, write, and execute. In file system rights management, read, write, execute, and no permissions are represented by four characters: r, w, x, and -. The numbers corresponding to the four rights are read permission: 4; write permission: 2; execution permission: 1; no permission: 0. The administrator sets access rights for
different users according to specific access requirements, which ensures that the damage caused by
the attacker's access to the normal account is minimized.

The DAC model establishes an access control list to implement rights management, centered on
user self-management. The MAC model is a security level that pre-defines user and data information,
and enforces allocation to implement rights management. RBAC separates the logical relationship
between the authority and the user, and introduces the concept of the role to implement the
management of the authority. The Linux system manages file permissions by setting permission bits.
Compared with other access control models, Linux UGO model can more clearly and intuitively view
each user's access to files, which is easier to manage permissions.

The above comments not only indicate that more research is needed in this area, but it also provides
insights and guidance for our research.

A Proposed Runtime Customizable Access Control Model

The Model Definition

The RBAC model is used as a basic study by comparing the advantages and disadvantages of the
above three control models. The RBAC model is a way for users to associate with roles. A specific
user may be authorized to one or several roles, and each role and its permissions are pre-defined at
the beginning of the system design, each role accords to several permissions. It is hard for managers
to dynamically define new roles for users with personalized permissions during system runtime after
system delivery.

Based on the RBAC model, we propose a CRBAC (customizable Role-Base Access Control)
model. The CRBAC model is designed for those project cooperation systems with many access
objects and fewer users, each user has different access requirements, and each user’s requirements
are uncertainty and can only be confirmed after a cooperation project has been launched.

Definition:

Let O be the objects set, \(O = \{o_1, o_2, ..., o_i, ..., o_m\}\), \(o_i\) represents an object or resource need be
authorized with access control permissions.

Let P be the operations set, \(P = \{p_1, p_2, ..., p_i, ..., p_n\}\), \(p_i\) represents an access operation such as
reading, writing, or execution.

Let R be the roles set, \(R = \{r_1, r_2, ..., r_j, ..., r_p\}\), \(r_i\in Power(O \times P)\), \(r_i\) represents a relationship of
role and its permissions.

Let U be the users set, \(U = \{u_1, u_2, ..., u_i, ..., u_q\}\), \(u_i\) represents a user.

Let A be the access control permissions set, \(A = \{a_1, a_2, ..., a_i, ..., a_r\}\), \(a_i\in Power(R \times U)\), \(a_i\)
represents a relationship between a role and its users.

In order to implement the runtime custom-built access control, a multi-to-one relationship is used
between U and R, each user has only one role. In the system runtime, if a special user’s access
requirements change, it is only necessary to change the role according to it or create a new role for it
if the current role has other users.

\[\forall u_i, \exists a_j (u_i \in U, a_j \in A, \text{and } u_i \in a_j)\]  \hspace{1cm} (1)

According to Eq. 1, The relationships between users, roles, and access control permissions are
shown as Figure 1.

![Figure 1. C-RBAC model diagram.](image-url)
In Figure 1, User represents the collection of users, Roles represents the collection of roles, and Permission represents all permission collections. Each user has only one role, and each permission belongs to at least one role. The roles are used to build relationship between users and permissions.

**Stored Method**

In the traditional RBAC, the user \( u_i \) first finds the role \( r_j \) through the association relationship between the role and the user and then finds the corresponding permission set \( \text{Power}(A) \) of the user through the association relationship between the role and the permission. When the authorization set \( A \) is of a large magnitude, each time we look up the permission \( q_i \) of \( u_i \), it has to go through a series of searches to find it. This type of search is a waste of time and inefficiency. At the same time, permission viewing is not intuitive and inconvenient.

In order to improve the efficiency of permission search, we refer to the Linux file permission definition to set the CRBAC model storage method, as shown in Figure 2.

![Figure 2. C-RBAC model storage method.](image)

In Figure 2, the access control permissions for each object \( o_i \) are designed as permission groups \( r_i \), each permission for \( o_i \) is tagged by a binary bit, and a linear array \( r[n] \), \( r[n] \in \text{Power}(O \times P) \), is used to storage all permissions to all objects for one role \( r_i \). All \( r[n] \) for all roles are items in \( R \).

When creating a role, we only need to assign a permission set array to it, and no longer need to associate different permissions with various roles. The user only needs to obtain the array management of the linear key permission set of the corresponding role and can find the corresponding authority, which greatly improves the query efficiency, saves time, and improves the system running speed.

**Access Control Permission Search Algorithm**

<table>
<thead>
<tr>
<th>Algorithm: permission search</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong>: A user ( u ), an access control permission array ( A ), a role set ( R ).</td>
</tr>
<tr>
<td><strong>Output</strong>: the permission of a role ( r_i ).</td>
</tr>
</tbody>
</table>

For each \( a_i \) in \( A \) // \( A \subseteq \text{Power}(R \times U) \)
if \( u = a_i.u \) Then \( r = a_i.r \)
For each \( o_i \) in current interface
For each \( p_j \) in \( r[i] \) // \( p_1, p_2, ..., p_n \) of \( o_i \) (See Figure 3)
If \( p_j = 1 \) Then present \( o_i \) has this privilege // Edit permission, Delete permission, etc.,
Else \( p_j = 0 \) Then present \( o_i \) does not have this permission

When a user \( u \) is given, the role \( r \) corresponding to it will be found with the help of the set \( A \). For each object \( o_i \) in current interface, check its access control binary bit tag in \( r[i] \). If the tag is 1 then present \( o_i \) has this permission according the predefined access control permission, such as Edit permission and Delete permission. Otherwise, present \( o_i \) does not have this permission.

When the user executes the transaction, the array of role permission sets is obtained, and the permission search algorithm can quickly obtain the user rights and obtain the corresponding permission data resources.
**An Example Explanation**

Based on the above model, we will give an example for explanation and explanation.

Assume that there are 4 types of access objects O, the distribution is o1, o2, o3, o4; there are 3 kinds of operation P, the distribution is p1, p2, p3; then the authorization set P has twelve permissions.

There are three types of character sets R, which are r1, r2 and r3. Through the authorization process of Figure 3, we get an array of permission sets for r1.

![Figure 3. r1 permission set array.](image)

In Figure 3, we assign the value of each group O by assigning each pi, and finally get the permission set array r[4].

According to the authorization process of Figure 4, we authorize the remaining roles separately and finally get the role set R shown in Table 2.

<table>
<thead>
<tr>
<th>Character</th>
<th>Privilege array</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>101011100110</td>
</tr>
<tr>
<td>r2</td>
<td>000000000000</td>
</tr>
<tr>
<td>r3</td>
<td>111111111111</td>
</tr>
</tbody>
</table>

In Table 2, the array of permission sets for r1, r2, r3 and their corresponding binary arrays are shown.

According to the binary number in Table 2, we get the permissions that each role has. As shown in Table 3.

<table>
<thead>
<tr>
<th>Character</th>
<th>O1</th>
<th>O2</th>
<th>O3</th>
<th>O4</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>r2</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>r3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

In Table 3, we can clearly see the permissions that each role has. (✓ indicates that this permission is available, , x indicates that this permission is unavailable).

When the user logs in, through the permission search algorithm, all the permissions owned by the user can be quickly found.

In this section, we compare the three access control models and select the RBAC model as the access control model. Based on the RBAC, we propose the C-RBAC model. Changed the traditional storage method, custom designed a storage method. Redefining the permission search method makes the storage of the C-RBAC model more convenient, and at the same time, the permission search is faster and more efficient. Permission viewing is more intuitive and convenient. And an example to illustrate the rights allocation method permission storage method and permission search method.
Case Study

To further validate the validity of the model, we implemented a rights management module in a garment design cooperation system. The system is used to help stakeholders to cooperate in garment design cooperation projects.

A project generally involves 26 kinds of categories (26 objects), each object has 3 kinds of access control permissions. That means that there are 78 access control permissions in this system. Generally, there are 5 kinds of stakeholders in a garment design cooperation projects: design director, fashion designer, clothing technologist, clothing pattern maker, and clothing sampler. A real project always has few users and a special user may have different access requirements according to the ability and division of labor of project members. A runtime customizable access control model is needed to manage the access control permissions for users with dynamic requirements. By using the system, the design director can manage users in three steps. First, create a role (see Figures 4 and 5); second, create a user (see Figure 6); finally manage the user (see Figure 7). With this system, administrators can authorize each user to have different access rights in different projects. At the same time, can customize the permissions of the same person.

![Figure 4. The role definition.](image)

In Figure 4, all the modules and corresponding operation rights of the system are listed, so that the administrator can conveniently view all the rights, and at the same time, dynamically manage and select the rights according to the corresponding roles.

After the administrator has set the role according to the role requirements, you can easily view and manage all the roles through Figure 5.

![Figure 5. Role management diagram.](image)
In Figure 5, the administrator can clearly view all the configured roles. At the same time, the roles can be dynamically managed and the role permissions can be modified and deleted at any time.

When the administrator assigns a different role to a different user according to the requirements, it can be viewed and managed through Figure 6.

![Figure 6. Add user graph.](image)

In Figure 6, the administrator can assign the created new user to the corresponding role according to the requirements, which is convenient for the administrator to manage the user.

After the user is created successfully, we can view all users and their information in Figure 7.

![Figure 7. User role diagram.](image)

In Figure 7, the administrator can clearly view the user and the role corresponding to each user. At the same time, the user can be dynamically managed, and the user role can be modified or deleted at any time. Make it easier for administrators to manage users.

In this section, the design implements a set of clothing design process rights management module. According to the requirements, the definition of roles and permissions is completed, and the dynamic management of users and roles is realized. This rights management module makes it easier and faster for administrators to manage system permissions and improve work efficiency.
Conclusion

Various custom-built cooperation software systems generally adopt role-based access control, and each role and its permissions are pre-defined at the beginning of the system design. This paper proposes a runtime customizable access control model for the custom-built cooperation software system. Through this model, the managers can customize roles and dynamically authorize them to project members in real time during the runtime of the system. At the same time, the model can view user permissions intuitively and conveniently. The validity of the model is verified by a clothing design cooperation system. The results show that with the use of this model, managers can dynamically create personalized roles and authorize them to different project members when a new cooperation project is launching. Simultaneously, administrators can easily and intuitively manage user rights. This model and the case can help relevant researchers and developers to design and develop their own runtime customizable access control model and custom-built cooperation software system. It is our intention to further carry out research to develop dynamic access control models and case studies that can be used to build ease of use and auditable cooperation system.

References


